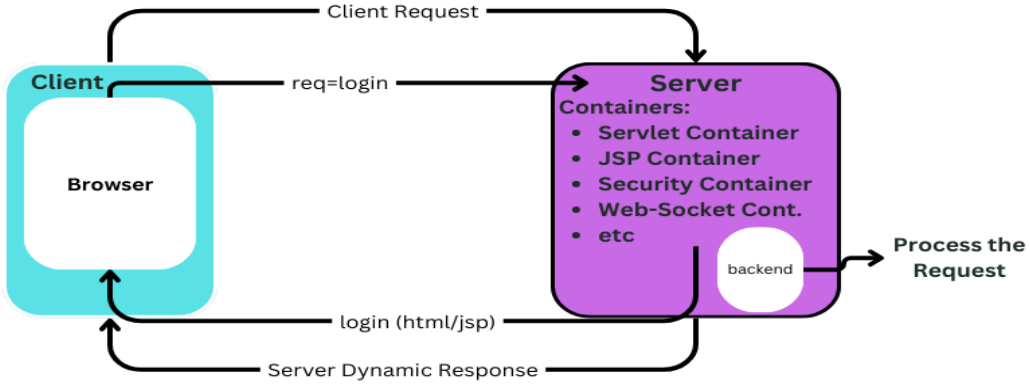
Servlet-Tutorial

Introduction to Servlet:

1. It is a Server-Side Technology.
2. A servlet is a Java program for web applications.
3. It handles HTTP requests and responses.
4. Servlets run on a server.
5. It has a defined lifecycle (init, service, destroy).
6. Used for generating dynamic web content.

Responsibilities to Servlet:

- Handles the client request.
- Process the Request.
- Generate Dynamic Response.



Client → Request → Server (Process Request**) →Generate→Dynamic Response→Client**

Server:

- A server provides resources and services to clients.
- It handles client requests and sends responses.
- Servers support web, database, and application functionalities.

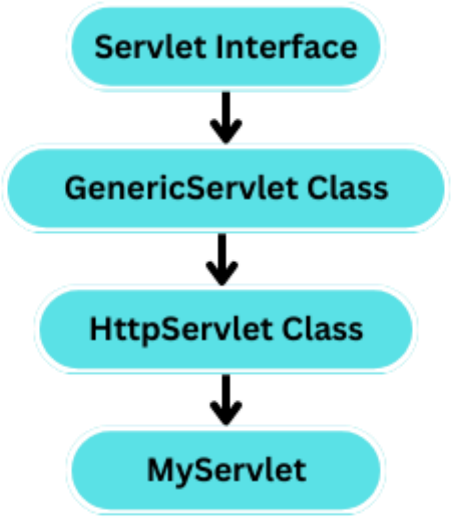
1. **Web Server :**
    a. Web servers manage HTTP requests for websites.
    b. Examples:
        1. Apache Tomcat
        2. GlassFish
        3. Jetty

2. **Application Server :**
    a. Application servers handle business logic and backend processing.
    b. Examples:
        1. JBoss
        2. Web Logic
        3. IBM Webspace

Steps to Add/Configure Tomcat Server:

1. Download Eclipse & Tomcat zip file.
2. Set the runtime environment of Tomcat in Eclipse.
3. Add the server in Eclipse.

Hierarchy of Servlet :



1. **Servlet Interface** :

    The root interface defining servlet methods.(Parent Interface)

2. **GenericServlet Class** :

    Implements the Servlet interface, providing protocol independent methods.

3. **HttpServlet Class** :

    Extends GenericServlet, specialized for handling HTTP requests.

4. **Custom Servlet Class** :

    User-defined class that extends HttpServlet to create a specific servlet.

Syntax :

1. class MyServlet implements Servlet
   {
   Backend Code
   }

2. class MyServlet extends GenericServlet
   {
   Backend Code
   }

3. class MyServlet extends HttpServlet
   {
   Backend Code
   }

Note : We have to create a Deployment Descriptor File. i.e web.xml

Life-Cycle of Servlet:

The lifecycle of a servlet consists of the following stages:

1. **Loading and Instantiation**:

   The servlet is loaded into memory and an instance is created.

2. **Initialization (`init()`)**:

   The servlet is initialized and configured once after instantiation.

3. **Request Handling (`service()`)**:

   For each client request, the `service()` method is called to process the request and generate a response.

4. **Destruction (`destroy()`)**:

   Before the servlet is removed from memory, the `destroy()` method is called to clean up resources.

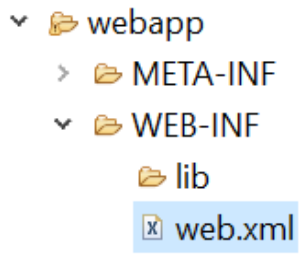
Note : When multiple request occurs, each request has its new own thread i.e(**MultiThreading**)

Deployment Descriptor (web.xml) :

- The **Deployment Descriptor (web.xml)** is an XML file used to configure servlets in a web application. It is located in the `WEB-INF` directory of the web app and defines settings like servlet mappings, initialization parameters, and other configurations.
- Key elements:
    - <servlet>: Defines a servlet by name and class.
    - <servlet-mapping>: Maps a servlet to a URL pattern.
    - <welcome-file-list>: Specifies default files (e.g., index.html) to be loaded when accessing the root of the application.
- Example :

```
<display-name>ServletApplication2</display-name>
<servlet-mapping>
        <servlet-name>servletname</servlet-name>
        <url-pattern>/abcurl</url-pattern>
</servlet-mapping>
<servlet>
        <servlet-name>servletname</servlet-name>
        <servlet-class>in.backend.MyServlet</servlet-class>
</servlet>
```

- Location :



- Different tasks of web.xml file :-
    1. Servlet Configuration
    2. JSP File Configuration
    3. Filters Configurations
    4. Listeners Configurations
    5. Error Page Configuration
    6. Welcome File Configuration

Note : In latest version, it is not necessary to create web.xml file.
Hint - @WebServlet Annotation

@WebServlet Annotation :

- **Defines servlet**: The `@WebServlet` annotation simplifies servlet definition without needing `web.xml`.
- **URL mapping**: It allows URL patterns to be mapped directly in the code.
- Syntax: @WebServlet("/url")

Note: Annotations i.e. @WebServlet, @WebFilter, @WebListener, @MultipartConfig etc are part of the Java Servlet API and can be used in any version from servlet 3.0 onwards, including newer versions like 3.1, 4.0, and 5.0

Introduction to HttpServletRequest and HttpServletResponse:

- Both are interfaces.

HttpServletRequest:

- Represents the client's request, containing data like parameters, headers, and method types.
- Methods :
    - **getParameter(String name):** Returns the value of the request parameter specified by the name.
    - **getCookies():** Returns an array of Cookie objects representing the cookies included in the request.
    - **getSession(boolean create)**: Returns the current session associated with the request or creates a new one if create is true.
    - **getMethod():** Returns the HTTP method of the request, such as GET, POST, PUT, DELETE, etc.
    - **getAttribute(String name):** Returns the value of the named attribute as an Object.
    - setAttribute(String name, Object value): Binds an object to a given attribute name in the request scope.
    - **getHeader(String name):** Returns the value of the specified HTTP header.
    - **getHeaderNames():** Returns an enumeration of all the header names sent with the request.

HttpServletResponse:

- Represents the client's request, containing data like parameters, headers, and method types.
- Methods :

    → **getWriter():** Returns a PrintWriter object that can be used to send character text to the client.

    → **setContentType(String type):** Sets the MIME type of the response.

    → **setContentLength (int len)**: Sets the length of the content being returned in the response.

    → **sendRedirect(String location):** Redirects the client to a different URL.

    → **sendError(int sc, String msg):** Sends an error response to the client with the specified status code and message.

    → **addCookie(Cookie cookie):** Adds a cookie to the response.

    → **setStatus(int sc):** Sets the status code of the response.

    → **setHeader(String name, String value)**: Sets the value of the specified response header.

    → **addHeader(String name, String value):** Adds a response header with the given name and value.

Http Methods :

- Some commonly used HTTP methods are:-
    - **GET**:- The GET method is used to retrieve information from the given server using a given URL. Requests using GET should only retrieve data and should have no other effect on the data.
    - **HEAD** :- Same as GET, but transfers the status line and header section only.
    - **POST**:- A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
    - **PUT** :- Replaces all current representations of the target resource with the uploaded content.
    - **DELETE** :- Removes all current representations of the target resource given by a URL.

1. **GET:**
    a. GET method sends data through the resource URL and thus it is not secured.
    b. GET is slightly faster because the values are sent in the header.
    c. We can send very less data in case of GET request because it adds the data to the URL and the length of a URL is limited (maximum URL length is 2048 characters)
    d. for GET type request --> use **doGet()** method
2. **POST:**
    a. POST method sends data through the HTTP message body and thus it is more secure.
    b. POST is slightly slow because the values are sent in the request body, in the format that the content type specifies.
    c. We can send a huge amount of data in case of POST request, there is no restriction.
    d. for POST type request --> use **doPost()** method.

Difference between GET and POST :-

GET	POST
● GET method sends data through the resource URL ● GET is not secured as data is visible in URL ● We can send very less data using GET request because data is transferred using URL ● GET request can be cached ● GET request can be bookmarked	● POST method sends data through the HTTP message body ● POST is more secured as data is not visible in URL ● We can send more data using POST because it does not send data using URL ● POST request cannot be cached ● POST method cannot be bookmarked

sendRedirect() and RequestDispatcher :

1. **sendRedirect():**
    a. sendRedirect() method is used to redirect the response to another resource (ie. to servlet or JSP or html etc)
    b. It is the method of HttpServletResponse.
    c. Example : response.sendRedirect("login.html");

2.
3. **RequestDispatcher :**
    a. The RequestDispatcher interface is used to dispatch the request to another resource (servlet or jsp or HTML within same application)
    b. It is called by res.getRequestDispatcher().
    c. Example: RequestDispatcher dispatcher = request.getRequestDispatcher("JSPprofile.jsp"); dispatcher.forward(request, response);

sendRedirect()	RequestDispatcher
• It is used for external request redirection. • It redirects the request to a different application or URL. • It is called by HttpServletResponse object. • Syntax:- ○ For any URL: response.sendRedirect("https://www.example.com"); ○ For any servlet : response.sendRedirect("/servlet2"); • It change the URL on the browser	• It is used for internal request redirection. • It forwards or includes the request to the same application or URL. • It has 2 methods, forward() and include(). • Syntax :- ○ forward() method: rd.forward(request, response); ○ include() method: rd.include(request, response); • It does not change the URL on the browser.

Http Session :

- **HttpSession**: Used to track user sessions across multiple requests in a web application.
- **Session management**: Stores user-specific data like login info, preferences, and state across interactions.
- **Unique ID**: Each session is assigned a unique session ID to identify user data.
- **Creation and destruction**: A session is created when a user first accesses the application and can be destroyed manually or by timeout.

Steps:

1. **Create session**: creates a new session or retrieves an existing one.

   → HttpSession session=req.getSession();

2. **Set attribute**: stores the `nameString` object with the key `"name_key"` in the session.

   → session.setAttribute("name_key", nameString);

3. **Get attribute**: retrieves the object associated with `"name_key"` and casts it to a `String`.

   → String nameString = (String)session.getAttribute("name_key");

4. **Remove attribute**: removes the attribute associated with `"name_key"` from the session.

   → session.removeAttribute("name_key");

5. **Invalidate session**: terminates the session, invalidating all stored attributes and data.

   → session.invalidate();